UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA
Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea in Informatica, corso Big Data

Relazione attività MongoDB

Studente:

Lorenzo Stigliano
Matricola 136174

Anno Accademico 2022-2023

Contents

1 Pre-requisiti

Di seguito i pre-requisiti, con rispettive guide, per poter utilizzare il usufruire del codice e del database che verranno successivamente proposti:

- MongoDB [1]
- Python 3.x [2]
- Pip [4]
- Pymongo [5]
- Git [3]

Tutto il codice a cui si fa riferimento è stato caricato all'interno della seguente cartella su *GitHub* [6]. La cartella ha 3 sotto-cartelle:

- JSON: contiene file *json* utilizzati come database per MongoDB e per effettuare un'operazione di *preprocessing* sui dati.
- Python: all'interno è presente il programma principale che contiene l'implementazione e il testing della libreria *PyMongo* e delle sue funzionalità, insieme al programma utilizzato per effettuare una modifica sul database che verrà esplicitata nei capitoli successivi.
- JupyterNotebook: contiene un documento JupyterNotebook contenente il programma principale con alcuni commenti.

2 MongoDB

2.1 Setup

Ai fini di questa relazione è stato utilizzato un database in formato *json* ottenibile tramite il seguente link: city_inspections_db.json. Questo file contiene 81k documenti, ciascuno dei quali identifica una ispezione delle autorità nei confronti di attività commerciali di varia natura presenti sul territorio americano. Questi documenti sono costituiti nel seguente modo:

- id: numero identificativo univoco della ispezione
- certificate_number: certificato univoco dell'ispezione
- business_name: nome dell'attività
- date: data della verifica
- result: risultato dell'ispezione
- sector: settore in cui opera l'attività
- address
  - city: città in cui risiede l'attività
  - zip: codice postale
  - street: indirizzo della attività
  - number: recapito telefonico

2.2 Modifiche al Database

Per poter interrogare correttamente il database è stato modificato il formato delle date di ogni documento, passando da: `Feb 20 2015` (Month-Day-Year) a `2015-2-20` (Year-Month-Day). Grazie a questo formato ora è possibile operatori quali: `gt`, `gte`, `lt`, `lte`, etc. ... per definire delle finestre temporali all'interno delle quali formulare le interrogazioni. Per effettuare tale modifica è stato utilizzato lo script in Python date_fix.py. All'interno di questo file troviamo 2 funzioni, oltre il `main`:

- `get_month`: dato un mese ne ritorna il corrispettivo numerico. Per esempio: dato *Feb* ottengo `2` come valore di ritorno;
- `FromStringToDate`: funzione che fornisce la conversione delle date dal formato *Month-Day-Year* a *Year-Month-Day*;

Queste due funzioni sono utilizzate all'interno del `main` per poter modificare le linee del documento `city_inspections.db`. Viene individuata la posizione della data attraverso una *regex* su ogni linea del documento e tramite il metodo `replace` viene modificato tale campo. Si ottiene quindi una nuova linea di testo che viene scritta nel file `city_inspections_db.json`.

```
def main():
    output = open("city_inspections_db.json", "w")
    with open('city_inspections.json', 'r+') as f:
        for line in f:

            result = re.search('"date":"(.*?)"', line)
            date = (result.group(1))

            x = str(line).replace(date , FromStringToDate(date), 1)
            output.write(x)


if __name__ == "__main__":

```

3 Python

3.1 PyMongo

Per poter effettuare delle interrogazioni in *Python* è stato utilizzata la libreria *PyMongo*. Questa ci fornisce tutti gli strumenti utili per poter interagire con il database, fornendo metodi per la connessione, modifica e inserimento di documenti e possibilità di effettuare interrogazioni.

Il file utilizzato come demo per l'utilizzo della suddetta libreria ha al suo interno cinque funzioni principali, oltre alla funzione `main`, che permettono le seguenti operazioni:

- `connect`: permette la connessione al database;
- `insert`: consente l'inserimento di un nuovo documento;
- `update`: rende possibile l'aggiornamento di un documento;
- `indexing`: permette di creare un indice su un campo dei nostri oggetti del database;
- `query`: fornisce capacità di interrogazione sul database;

3.1.1 Connect

La sintassi della funzione `connect` fa riferimento alla documentazione di *PyMongo* e viene definita come segue:

```
1 server = "mongodb://localhost:27017/"
2 database = "City"
3 collection = "City_Inspections_DB"
4
5 def connect():
6   myclient = pymongo.MongoClient(server)
7   mydb = myclient[database]
8   mycol = mydb[collection]
9   return mycol
```

Come è possibile vedere si utilizzano i dati forniti all'inizio del file per connettersi al server locale in cui è contenuto il database. Editando questi campi è possibile configurare l'accesso per altri server o database. La variabile *mycol* verrà ampiamente utilizzata in seguito per poter effettuare operazioni all'interno della collezione di documenti utilizzata per questa relazione (`city_inspections_db.json`).

3.1.2 Insert

Per quel che concerne la funzione `insert` dobbiamo creare un dizionario con all'interno i valori desiderati e con il metodo *insert_one* aggiungere il documento appena creato al database:

```
1
2  def insert(mycol):
3
4    inspection = {"id" : "00000-0000-ENFO",
5    "certificate_number" : "17041999",
6    "business_name" : "Videogames Center",
7    "date" : "1999-4-17",
8    "result" : "Fail",
9    "sector" : "Videogames",
10   "address" : {
11     "city" : "Georgia",
12     "zip" : "41030",
13     "street" : "Rue de Baptiste",
14     "number" : "19"}}
15
16   check = mycol.find({"id":str(inspection.get("id"))}).distinct("id")
17
18   if not check:
19     mycol.insert_one(inspection)
20   else:
21     print("There is already this object")
```

Prima di aggiungere il documento al database è bene verificare che non esista già un documento con lo stesso identificativo univoco: `id`. Per poter effettuare tale controllo si fa uso di una interrogazione che controllo che cerca tra tutti i documenti quelli con l'identificativo dell'oggetto che stiamo per inserire e li salva, se ce ne sono, su una lista chiamata `check`. Se tale lista risulta essere vuota, allora possiamo procedere all'inserimento del nostro nuovo documento, perché vorrà dire che non è mai stato inserito, altrimenti non si procede con l'inserimento perché vuol dire che tale oggetto è già presente all'interno della raccolta.

3.1.3 Update

La funzione di `update` permette di modificare un documento della raccolta selezionandolo a partire dal suo id come segue:

```
1
2  def update(mycol):
3    query = {"id":"00000-0000-ENFO"}
4    new_values = {"$set":{"Inspector Name":"Lorenzo Stigliano"}}
5
```

```
  check = mycol.find(query).distinct("id")
  if not check:
    print("There is document with the right id to be updated")
  else:
    mycol.update_one(query, new_values, upsert = True)


```

Si crea una variabile *query* che definisce il parametro utilizzato per identificare l'oggetto sul quale vogliamo fare un aggiornamento, dopodiché si definisce il nuovo valore da aggiornare o inserire. In questo caso vogliamo inserire un nuovo valore, ovvero: il nome di chi esegue l'ispezione, ed è quindi necessario utilizzare l'operatore `$set`. A questo punto chiamo il metodo `update_one`, perché stiamo aggiornando solamente un documento, passando le variabili appena create come parametri e impostanto l'opzione `upsert` (*update* and *insert*) su `True`. In maniera simile alla fase di *insert* anche in questo caso si effettua un controllo per verificare che sia presente il documento su cui vogliamo fare l'update; se così non fosse si andrebbero a creare dei documenti aventi solamente il campo *id* e quello *Inspector Name*, con i valori indicati all'interno di `query` e `new_values`.

3.1.4 Indexing

La funzione di indexing permette di creare un indice per velocizzare le interrogazioni. Dato che sono ricorrenti le interrogazioni che utilizzano degli operatori che lavorano sul campo delle date, è stato deciso di utilizzare le date come indice. Per poter effettuare tale operazione è sufficiente la seguente riga di codice *Python*:

```

def index(mycol):
  mycol.create_index([('date', pymongo.ASCENDING )], name = index_name)

```

`mycol` rappresenta la collezione che viene utilizzata e `index_name` rappresenta il nome dell'indice che si intende utilizzare. I primi parametri all'interno del metodo `create_index` rappresentano rispettivamente: il campo sul quale viene creato l'indice e l'ordine dell'indice.

La libreria *PyMongo* implementa anche un metodo per poter eliminare un indice:

```

mycol.drop_index(index_name)

```

3.1.5 Query

Per testare le funzionalità di ricerca sono state create alcune interrogazioni demo. Nel primo caso si immagina di volere sapere in un dato periodo di tempo

quante ispezione abbiano dato esito negativo, facendo riferimento alla città di New York e alla via Frederick Douglass Buolevard. Per ottenere tale risultato si è strutturata l'interrogazione nel modo seguente:

```
result = mycol.find({"date" :
{"$gte":"2015-1-1", "$lte":"2015-6-30"},
"result" : {"$in" : ["Fail", "Violation
Issued"]},
                     "$and" : [{"address.city" :
                     "NEW YORK"},
                     {"address.street" :
                     "FREDERICK DOUGLASS  BLVD"}]}).sort("business_name").
                     distinct("business_name")
```

viene effettuata una interrogazione all'interno di una finestra temporale specifica, grazie agli operatori `gte` e `lte`, che definiscono il periodo che intercorre tra 2015-1-1 ed il 2016-6-30 per il campo *date*. Successivamente si cercano i documenti il cui risultato sia *Fail* o *Violation Issued*, grazie all'operatore `in` che identifica uno due valori all'interno del campo *result*. Vengono inoltre filtrati i risultati sui campi *address.city* e *address.street* per isolare solamente le indagine a New York in Frederick Douglass Boulevard. Il risultato finale viene poi ordinato in base al nome della attività e vengono stampati a schermo solamente questi ultimi.

Per quel che riguarda la seconda query è stato immaginato uno scenario in cui voglia effettuare una ricerca su due città diverse ed una singola tipologia di attività; questo al fine di ottenere l'ordine alfabetico di uno specifico tipo di attività in diverse locazioni. All'inizio si effettua un match, con l'operatore omonimo `match`, sugli zipcode e i settori delle attività (rispettivamente i campi: `address.zip` e `sector`), al fine di isolare le due città:

- NEW YORK
- ELMHURST

ed il settore: *Mobile Food Vendor - 881*. Isolati questi elementi si crea un gruppo con l'omonimo operatore, `group`, che contiene l'id dell'oggetto all'interno del database e il nome dell'attività. Si effettua infine una operazione di analisi dell'array (letteralmente sarebbe uno *srotolamento* dei campi), con l'operatore `unwind`, e viene effettuato un ordinamento ascendente con `sort` (passando l'argomento 1 insieme al campo su cui effettuare l'ordinamento) sui nomi delle attività.

```
query = [
  {"$match" : {"address.zip" : {"$in" : [10030, 11373]}}},
  {"$match" : {"sector" : "Mobile Food Vendor - 881"}},
  {"$group" : {'_id': '$id',
  'inspected_business': {'$addToSet': '$business_name'}}},
  {"$unwind" : "$inspected_business"},
```

```
        {'$sort': {'inspected_business': 1}}
    ]
```

Per la terza interrogazione proposta si considera l'eventualità di voler conoscere il numero delle ispezioni per indirizzo stradale, di una singola città e considerando solamente una tipologia di settore. Per fare ciò viene effettuato un match su tre campi:

- `address.zip`: 11234 (BROOKLYN);
- `sector`: Cigarette Retail Dealer - 127;
- `date`: dal 2016-1-1 al 2016-4-30;

Dopodiché viene creato un gruppo all'interno del quale sono indicati l'identificativo dell'oggetto (`id`) e la via in cui avviene l'ispezione (`address.street`). Si effettua una analisi del gruppo appena creato e viene fatta una somma delle ispezioni per ogni via grazie all'operatore `sum` mentre si crea un gruppo in cui sono indicati gli identificativi e gli indirizzi delle città in cui sono stati effettuati i controlli. Infine si ordina in maniera ascendente rispetto al numero di controlli per ogni via.

```
    query = [
      {"$match" :
          {"address.zip" : 11234,
          "sector" : "Cigarette Retail Dealer - 127",
          "date" : {"$gte":"2016-1-1", "$lte":"2016-4-30"}}},
      {"$group" : {
          "_id" : "$id",
          "street_inspected" : {"$addToSet" : '$address.street'}}},
      {"$unwind" : "$street_inspected"},
      {'$group': {'_id': '$street_inspected', 'count': { '$sum': 1}}},
      {'$sort': {'count': 1}}
    ]


```

Per l'ultima interrogazione è stato ipotizzato uno scenario in cui si volesse conoscere il tipo di attività che avesse il numero più alto di *Fail* o *Violation Issued* come risultati dell'ispezione, considerando tre diversi quartieri di New York:

- Bronx
- Brooklyn
- Queens

ed il periodo che intercorre tra il 2015-1-1 ed il 2016-12-31. Di seguito l'implementazione della suddetta interrogazione:

```
query = [
  {"$match" : {"address.zip" : {"$in" : [10475, 11234, 11427]},
  "result" : {"$in" : ["Fail", "Violation Issued"]},
  "date" : {"$gte":"2015-1-1", "$lte":"2016-12-31"}}},
  {"$group" : { "_id" : "$certificate_number",
  "sector_inspected" : {"$addToSet" : '$sector'}}},
  {"$unwind" : "$sector_inspected"},
  {'$group': {'_id': '$sector_inspected', 'count': { '$sum': 1}}},
  {'$sort': {'count': -1}},
  {'$group': { '_id': "$sector_inspected", 'maxval': { '$first': '$$ROOT'}}},
  {'$replaceWith': '$maxval' }
]
```

Per prima cosa è stato effettuato un match per i quartieri di New York, i due risultati di interesse ed il periodo di riferimento. Fatto ciò è stato creato un gruppo che comprende il numero del certificato della ispezione degli oggetti trovati ed il settore di appartenenza. Si procede con una ispezione del settore di appartenenza ed un conteggio di dei documenti interessati per ogni settore, per poi ordinare in modo decrescente il risultato. Si effettua infine una selezione sul primo risultato ottenuto attraverso la creazione di un gruppo da cui si estrae solamente il primo elemento.

References

[1] Install MongoDB, https://www.mongodb.com/docs/manual/installation/

[2] Download Python, https://www.python.org/downloads/

[3] Installing Git, https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

[4] Installing Pip, https://pip.pypa.io/en/stable/installation/

[5] Download pymongo, https://pypi.org/project/pymongo/

[6] Demo MongoDB, https://github.com/MaxBubblegum47/Demo_MongoDB